What is Multi Core Architecture?

When a processor has more than one core to execute all the necessary functions of a computer, it’s processor is known to be a multi core architecture.

In other words, a chip with more than one CPU’s (Central Processing Unit).
What is the difference between a single core processor and a multi core processor?
Advantages of multi core CPU

The largest boost in performance will likely be noticed in improved response-time while running CPU-intensive processes, like anti-virus scans, ripping/burning media.

Assuming that the die can fit into the package, physically, the multi-core CPU designs require much less printed Circuit Board (PCB) space than multi-chip SMP designs. Also, a dual core processor uses slightly less power than two coupled single core processors, principally because of the decreased power required to drive signals external to the chip.
Why Parallelization?

Parallelization is another optimization technique. The goal is to reduce the execution time.

To this end, multiple processors, or cores, are used.

Using 4 cores, the execution time is 1/4 of the single core time.
What Is Parallelization?

"Something" is parallel if there is a certain level of independence in the order of operations.

In other words, it doesn't matter in what order those operations are performed.

- A sequence of machine instructions
- A collection of program statements
- An algorithm
- The problem you're trying to solve
What is a Thread?

- Loosely said, a thread consists of a series of instructions with its own program counter ("PC") and state.
- A parallel program executes threads in parallel.
- These threads are then scheduled onto processors.
OpenMP

Shared Memory

http://www.openmp.org
What is OpenMP?

- De-facto standard API for writing shared memory parallel applications in C, C++, and Fortran

- Consists of:
  - Compiler directives
  - Run time routines
  - Environment variables

- Specification maintained by the OpenMP Architecture Review Board (http://www.openmp.org)

- Version 3.0 has been released May 2008
Advantages of OpenMP

- Good performance and scalability
  - If you do it right....
- De-facto and mature standard
  - Supported by a large number of compilers
- Requires little programming effort
- Allows the program to be parallelized incrementally
Shared Memory Model

All threads have access to the same, globally shared, memory.
Data can be shared or private.
Shared data is accessible by all threads.
Private data can only be accessed by the thread that owns it.
Data transfer is transparent to the programmer.
Synchronization takes place, but it is mostly implicit.
The OpenMP Execution Model

Fork and Join Model

Master Thread

Parallel region

Worker Threads

Synchronization

Parallel region

Worker Threads

Synchronization
Bio-grid Cluster

<table>
<thead>
<tr>
<th>Qty</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Head Node (x3650)</td>
</tr>
<tr>
<td>11</td>
<td>Compute Nodes (X3 250)</td>
</tr>
<tr>
<td>1</td>
<td>Std Rack</td>
</tr>
<tr>
<td>1</td>
<td>PDU for Racks (4)</td>
</tr>
<tr>
<td>1</td>
<td>Rack &amp; Install OS</td>
</tr>
<tr>
<td>1</td>
<td>Cisco Switch</td>
</tr>
</tbody>
</table>
How it works!!!
OpenMP example

• When writing a program in C or C++, you must import the library #include<omp.h> to include the openMP library on your program to allow parallelization during compile time.

• some sample OpenMP runtime Functions:
  - omp_set_num_threads (Sets number of threads)
  - omp_set_max_active_levels (Sets number of active parallel regions)
OpenMP Example Continued..

- `#include <omp.h>`
- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#define NRA 62` /* number of rows in matrix A */
- `#define NCA 15` /* number of columns in matrix A */
- `#define NCB 7` /* number of columns in matrix B */
int main (int argc, char *argv[ ])  
{
  int tid, nthreads, i, j, k, chunk;
  double a[NRA][NCA],       /* matrix A to be multiplied */
      b[NCA][NCB],       /* matrix B to be multiplied */
      c[NRA][NCB];       /* result matrix C */
  chunk = 10;               /* set loop iteration chunk size */
OpenMP Example Continued..

- /*** Spawn a parallel region explicitly scoping all variables ***/
- #pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
- {
-   tid = omp_get_thread_num();
-   if (tid == 0)
-     {
-       nthreads = omp_get_num_threads();
-       printf("Starting matrix multiple example with %d threads\n",nthreads);
-       printf("Initializing matrices...\n");
-       printf("Hello");
-     }
OpenMP Example Continued..

- /*** Initialize matrices ***/
- #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCA; j++)
      a[i][j]= i+j;
- #pragma omp for schedule (static, chunk)
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j]= i*j;
- #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      c[i][j]= 0;
/** Do matrix multiply sharing iterations on outer loop ***/
/*** Display who does which iterations for demonstration purposes ***/
printf("Thread %d starting matrix multiply...\n",tid);
#pragma omp for schedule (static, chunk)
for (i=0; i<NRA; i++)
{
    printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<NCB; j++)
        for (k=0; k<NCA; k++)
            c[i][j] += a[i][k] * b[k][j];
}
}   /*** End of parallel region ***/
/// Print results ///
printf("**************************************************\n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++)
{
    for (j=0; j<NCB; j++)
        printf("%6.2f   ", c[i][j]);
    printf("\n");
}
printf("**************************************************\n");
printf ("Done.\n");
}
The relevance...

Using the algorithm derived from solving the Matrix multiplication problem using OpenMP, we can extend this algorithm to solve computational biology problems. Such as

- “developing Algorithms for Identifying Boolean Networks and Related Biological Networks Based on Matrix Multiplication and Fingerprint Function”...
The relevance continued...

By doing so we can analyze gene expression data. Understanding matrix multiplication problem and it can be applied to solve cient algorithms for identifying Boolean networks of bounded indegree and related biological networks, where identification of a Boolean network can be formalized as a problem of identifying many Boolean functions simultaneously.”1
How to extend my project...

- Add MPI library and functions to send task to different processors
- Extend this algorithm to solve computational biology problems as stated on previous two slides.
Acknowledgements

- Special thanks to NSF for providing us with this wonderful opportunity to do research under BioGrid
- Also, Special Acknowledgements to Dr. Chun-His Huang for all his effort, patience and help towards helping me understand and implement the contents of this research!
- It was a fun summer indeed 😊
Source Citation

1. JOURNAL OF COMPUTATIONAL BIOLOGY
   Volume 7, Numbers 3/4, 2000
   Mary Ann Liebert, Inc.
   Pp. 331–343
   Algorithms for Identifying Boolean Networks and
   Related Biological Networks Based on Matrix
   Multiplication and Fingerprint Function
   TATSUYA AKUTSU, SATORU MIYANO, and SATORU KUHARA
2. wikipedia.org
Source Citation

- http://openmp.org/wp/
- https://iwomp.zih.tudresden.de/downloads/2.Overview_OpenMP.pdf